METHOD, APPARATUS AND SOFTWARE FOR PREVENTING SWITCH FAILURES IN THE PRESENCE OF FAULTS

FIELD OF THE INVENTION

The present invention is related to an electronic device having a master/slave bus interconnecting one or more bus master units with one or more bus slave units. More specifically, the present invention is related to a switch having a master/slave bus that automatically recovers from failures of one or more bus slave units based upon the operation of a software program.

BACKGROUND OF THE INVENTION

Systems based, in part, upon Master/Slave shared bus systems are vulnerable to system failure in the presence of certain kinds of slave unit failures. Microsoft windows implements detection of these failures but does not have the ability to recover the system. The present invention allows systems to recover from these failures without special purpose redundancy and resiliency hardware support. The present invention, utilizing software modifications, renders switching systems invulnerable to system failure in the presence of hardware faults in slave units. This method requires no special hardware support.

SUMMARY OF THE INVENTION

The present invention pertains to a switch for transferring data. The switch comprises at least one master unit. The switch comprises a plurality of slave units. The switch comprises a bus through which the master unit communicates with the slave units. The switch comprises a memory in communication with the master unit having a software program which causes the switch to automatically recover when a slave unit fails.

The present invention pertains to a method for transferring data. The method comprises the steps of attempting to access a failed slave unit of a plurality of slave units of a switch by a master unit of the switch with a signal through a bus through which the master unit and the failed slave unit communicate. There is the step of automatically recovering the switch from the failed slave unit with a software program in the switch that directs the master unit to avoid further accessing the failed slave unit of the plurality of slave units.

The present invention pertains to a software program. The software program comprises the steps of identifying a first slave unit of a plurality of slave units of a switch has failed when the first slave unit is attempted to be accessed by a master unit of the switch. The software program comprises the step of preventing a master unit from attempting to access the failed first slave unit.

BRIEF DESCRIPTION OF THE DRAWINGS

In the accompanying drawings, the preferred embodiment of the invention and preferred methods of practicing the invention are illustrated in which:

Figure 1 is a flow chart of the present invention.

Figure 2 is a block diagram of a switch of the present invention.

DETAILED DESCRIPTION

Referring now to the drawings wherein like reference numerals refer to similar or identical parts throughout the several views, and more specifically to figures 1 and 2 thereof, there is shown a switch 10 for transferring data. The switch 10 comprises at least one master unit 12. The switch 10 comprises a plurality of slave units 14. The switch 10 comprises a bus 16 through which the master unit 12 communicates with the slave units 14. The switch 10 comprises a memory 18 in communication with the master unit 12 having a software program 22 which causes the switch 10 to automatically recover when a slave unit 14 fails.

Preferably, the switch 10 includes persistent storage 20 that survives across abnormal termination of the switch 10. The switch 10 preferably includes a mechanism for detecting failures of the slave units 14 and thereupon causes the switch 10 to abnormally terminate. Preferably, the software program 22 causes the switch 10 to automatically recover when the detecting mechanism causes the switch 10 to abnormally terminate. The detecting mechanism preferably includes a hardware watchdog device 27.

The present invention pertains to a method for transferring data. The method comprises the steps of attempting to access a failed slave unit 14 of a plurality of slave units 14 of a switch 10 by a master unit 12 of the switch 10 with a signal through a bus 16 through which the master unit 12 and the failed slave unit 14 communicate. There is the step of automatically recovering the switch 10 from the failed slave unit 14 with a software program 22 in the switch 10 that directs the master unit 12 to avoid further accessing the failed slave unit 14 of the plurality of slavery units. Preferably, the recovering step includes the step of obtaining status information about the slave units 14 from persistent storage 20.

Referring to figure 1, the present invention pertains to a software program 22. The software program 22 comprises the steps of identifying a first slave unit 14 of a plurality of slave units 14 of a switch 10 has failed when the first slave unit 14 is attempted to be accessed by a master unit 12 of the switch 10. The software program 22 comprises the step of preventing a master unit 12 from attempting to access the failed first slave unit 14.

Preferably, there is the step of determining the switch 10 abnormally terminated when the master unit 12 attempted to access the first slave unit 14. There is preferably the step of changing information in persistent storage associated with the first slave unit 14 from identified as failed to identified as good if the switch 10 does not terminate abnormally after the master unit 12 attempts to contact the slave unit 14. Preferably, there is the step of setting a variable slot 24 chosen from amongst a plurality of slots 26 of the switch 10 not marked as potentially bad. There is preferably the step of determining whether the first slave unit 14 is physically present in a first slot 26 of the plurality of slots 26.

Preferably, there is the step of determining the first slot 26 is marked to be skipped. Preferably, there is the step of marking the variable slot 24 as potentially bad if it is not marked potentially bad. Preferably, there is the step of reporting the variable slot 24 as containing broken hardware and preventing the master unit 12 from attempting to access the variable slot 24 if the variable slot 24 is marked to be skipped.

There is preferably the step of attempting to access hardware present in the variable slot 24 if the variable slot 24 is marked potentially bad. Preferably, there is the step of marking the variable slot 24 as good if the switch 10 did not abnormally terminate when the master unit 12 accessed the first slave unit 14. There is preferably the step of enabling normal operations on hardware present in the variable slot 24 if the variable slot 24 is marked as good. Preferably, there is the step of setting the variable slot 24 to a next slot 26 of the plurality of slots 26.

In the preferred embodiment, persistent information is stored in a persistent storage 20 device, like a file system. This information is used to track the state of a slave hardware unit. If this slave unit 14 fails and causes the system to fail, the stored information is used subsequently to mark the hardware as suspect thus avoiding future hardware accesses to the failed slave unit 14 that may cause system failure. The attached flowchart illustrates this procedure in detail.

In the operation of the invention, the architecture of the switch is as follows.

Switch

a. Master/Slave Bus

b. Plurality of slave units

c. Control Processor

    i. Memory

        1. having a software program

    ii. Watchdog

    iii. Persistent Storage

    iv. Master Unit

The switch 10 comprises one or more control processors. Each control processor element contains memory 18 having a software program 22 that controls the switch 10, a watchdog device capable of detecting certain kinds of failures and also capable of restarting the switch 10 should a failure be detected, persistent storage 20 that retains information across system restarts and across loss of power to the system, and a master unit 12 which can instigate communications over a master/slave bus 16 to one or more slave units 14. Each of these components will be discussed in further detail in the following paragraphs.

The master slave bus 16 is used to interconnect units in a hardware system. It consists of an electrical interconnection between the various units and protocols that describe how the signals transported across the electrical interconnection are to be used to facilitate communications between the units attached to the bus 16.

Units attached to the bus 16 can be divided into two categories, master units 12 and slave units 14. Master units 12 are capable of initiating communications with other bus 16 units, while slave units 14 simply respond to communications initiated by the master units 12. A typical read transaction over the bus 16 starts with a master unit 12 making a request for information from a slave unit 14. The slave unit 14 accepts the requests, does whatever processing it needs to do locally to generate the information, and then returns the requested information and then the slave unit 14 signals that it has completed the transaction. A typical write request starts with the master unit 12 sending a write request to a slave unit 14 along with the data to be written. The slave unit 14 acknowledges the transaction, does whatever local processing is needed to process the write request, and then acknowledges the completion of the transaction.

The control processor subsystem manages the normal operation of the system and also contains a software algorithm that recovers the system automatically in the event of slave unit 14 failure. The control processor subsystem also contains memory 18 to hold the program and variables used by the program to help manage and recover the system. It also contains persistent storage 20 so that information may be retained across system restart events or system power down events. Finally, the control processor system has a watchdog mechanism. The purpose of the watchdog is to detect and recover from certain kinds of failures in the system.

Generally, a watchdog device operates by monitoring another device for activity. If the monitored device is inactive for too long a period, then it performs some action to recover the system. In the ASX-4000, the watchdog monitors the control processor's instruction fetch state. If the control processor stops accessing instructions for a long enough time period, the ASX-4000 watchdog resets the control processor subsystem, which instigates a system restart event.

Referring to figure 1, numbers within this text refer to specific boxes in the flow chart of figure 1 describing the software program. The text is organized as a walkthrough of the flowchart.

In the preferred embodiment, the present invention is implemented in Marconi's ASX-4000 switch 10 product. The ASX-4000 without the invention has been publicly available for purchase from before the filing date hereof. In this particular switch 10, the master-slave bus 16 connects the control processor to each of the slots 26 in the system. Each slot 26, when occupied, contains a slave device that is accessed by the control processor. The control processor acts as the master in the system. The key operational requirement of the control processor is that it is able to access a file system and, most importantly, the file system is capable of storing information in a synchronous manner, i.e. without buffering. This will be discussed more fully below.

The operation of the invention starts in the box labeled "1". Control proceeds to box "2" where a variable, named "SLOT" is initialized to point at the first slot 26. In general, the algorithm operates by iterating over all slave devices in the system. In the case of the ASX-4000, slave devices equate to system slots 26, hence, the flowchart refers to slots 26.

The first decision is made at the decision point box labeled "3" in the flowchart. The key point here is that once malfunctioning slave devices are removed from the system and are replaced with operational hardware, the persistent state that tracks the operational state of the hardware must be reset. If this were not done, the replaced hardware would continue to be treated as malfunctioning by the invention. If the slot 26 is empty, say because malfunctioning hardware was removed, control passes to the box labeled "10". Here, the slot 26, or in general the slave device, is marked as "absent" by placing an indication in the file system associated with the control processor.

If, at decision point "3", the slot 26 is found to be occupied, then control proceeds to the next decision point in the flowchart, labeled "4". The key point here is to check the file system to see if the slot 26 or slave device has already been judged to be "non operational" prior to the system being restarted. If so, control transfers to box "11", no future attempts are made to access the failed slave device, and appropriate action is taken to notify the operator of the system that a slave device in the system has been taken out of service. Control then passes to box "15" which runs the algorithm on any other unconsidered slave devices in the system. If all slave devices in the system have been tested, then the algorithm terminates normally as indicated by the transfer of control to box "16".

Returning to decision box "4", if the slave device being considered has not been marked as "to be skipped" during a previous invocation of the algorithm because the slave hardware is non-operational, then the algorithm proceeds to test the hardware. The algorithm checks to see if the hardware is marked as "potentially bad" at decision point 7 of the flowchart. If the slave device failed, and failure of a slave device causes the control processor to fail, during a previous invocation of the algorithm, the slave device would have been marked as "potentially bad" in the file system. Any slave devices marked as "potentially bad" in the file system must have caused the system to fail, so these devices are marked as "to be skipped" in box "6" of the flowchart.

However, if, at decision point "4", the hardware was not marked as "potentially bad", then the algorithm attempts to test the hardware by accessing the slave device. First, it marks the device as "potentially bad" in the file system. The method of marking is critical to the functioning of the algorithm. The file system must complete the write operation and have the information stored persistently BEFORE the device is accessed in box "9". Generally, the way to accomplish this is to invoke some sort of synchronize operation on the file system. For systems based upon Linux or other POSIX compliant operating systems, the fsync() system call accomplishes this. Marconi's implementation of this algorithm in Marconi's ForeThough software uses VxWorks ioctl operation to force synchronization of the entire file system. If the write does not complete before the slave device is accessed, the algorithm cannot recover from non-operational slave devices as the algorithm cannot track the failing slave device across invocations without the completion of this write operation.

Once the system has been marked, as shown in box "8", the system attempts to access the slave device as shown in box "9". If the device is operational, then the slot 26 is marked as good as shown in box "9" and control proceeds to box "14" to enable normal operations on the device and then to box "15" to see of other devices need to be checked.

If, on the other hand, the slave device is not operational, then the system will hang when the control processor attempts to manipulate the slave device. Eventually, hardware watch dog timers will detect that the system has failed and will restart the system. In this case the algorithm restarts and when control transfers to decision point "7", the failed hardware will be detected because of the information left in the file system during the previous invocation of this algorithm. This is how failing hardware is detected and marked as non-operational.

Eventually, all of the slave devices are checked and marked as either operational or non-operational. Once this happens, the algorithm terminates at box "16".

In any master/slave bus 16 devices attached to the bus 16 can be considered as either bus 16 masters or bus 16 slave devices. Master devices have the capability to initiate a transaction across the bus 16, while slave units 14 do not initiate any activity except when requested to do so by a master. In the preferred embodiment, the system control processor is the bus 16 master and all of the portcards act as slave devices.

A transaction on a master/slave bus 16 starts by a master unit 12 making a request of one of the slave units 14. The slave unit 14 either accepts or rejects the request, performs whatever actions it needs to do to satisfy the request, and then returns the result of the request to the master unit 12. Meanwhile, during the time it takes the slave unit 14 to perform the request, the master unit 12, the software program 22, and the system control processor 29, just wait. During this waiting period, the master unit 12, and the entire system 10, is essentially "frozen". Normally, this period is very small, on the order of a millionth of a second.

The problem is that certain hardware faults can cause the slave unit 14 to accept the request, and then fail while processing the request, leaving the master unit 12 and the entire system 10 permanently "frozen". There is hardware in the switch 10, called a watchdog, that detects if the master is frozen for too long and then performs a reset operation on the master unit 12. The phrase "abnormal termination" in the preferred embodiment refers to a bus 16 transaction that is terminated by having the watchdog hardware reset the master device instead of having the bus 16 transaction complete by having the slave device return the transaction result back to the master.

Persistent storage in Marconi's ATM switches such as the ASX-4000 is implemented using a flash file system. This is a solid state device, attached to the processor card, that appears to be a standard formatted file system. Similar devices are used in digital cameras. This store is manipulated by the program running on the processor via the VxWorks operating system's file system routines. The key enabler for the resiliency feature is that VxWorks supports transaction-like processing through some of the VxWorks system calls. This is detailed in the discussion of the flow chart above.

Although the invention has been described in detail in the foregoing embodiments for the purpose of illustration, it is to be understood that such detail is solely for that purpose and that variations can be made therein by those skilled in the art without departing from the spirit and scope of the invention except as it may be described by the following claims.